SIMULATING SENSOR NETWORKS IN NS-2 [1ST DRAFT]

Ian Downard
Naval Research Laboratory
Code 5523
4555 Overlook Ave
Washington DC, 20375-5337
downard@itd.nrl.navy.mil

May 21, 2003

Abstract

Building an optimal sensor network involves deliberately addressing each of a wide range of issues, such as physically detecting phenomena, deducing environmental information from raw sensor data, and communicating important alerts through an ad-hoc wireless network to an outside observer; all under tight energy constraints. Because of the number and complexity of these issues, the simulation environment provides an attractive universe where sensor network engineers can easily investigate the consequences of using various protocols and configurations. This was the motivation behind NRL's sensor network extensions to ns-2. In this paper, we describe how we added these extensions into the ns-2 framework, and illustrate their utility with a case study examining Mobile Ad-Hoc Network (MANET) routing within a dynamic sensor network. Finally, we will describe the limitations inherent to our simulation environment. Some sensor network related features in ns-2 are not compatible with our extensions, but as ns evolves, so do our on-going attempts at using that framework to accurately model sensor networks.

1 Introduction

Our idea of a sensor network is an autonomous multi-hop wireless network with nondeterministic routes over a set of heterogeneous physical layers. Our purpose is to evaluate how well current routing layer standards support the requirements of various sensor network applications.

The ns-2 simulation platform offers great flexibility to investigate the characteristics of sensor networks because it already contains flexible models for energy constrained wireless ad-hoc networks. In the ns-2 environment, a sensor network can be built with many of the same set of protocols and characteristics as those available in the real world. The mobile networking environment in ns-2 includes support for each of the paradigms and protocols show in Figure 1. In addition to the elements shown in Figure 1, the wireless model includes support for node movements and energy constraints. By leveraging the existing mobile networking infrastructure, we added the capability to simulate sensor networks. The only fundamental aspect of sensor networks missing in ns-2 was the notion of a "phenomenon channel" through which sensors could be triggered. Once a sensor detects the “ping” of a phenomenon in that channel, the sensor acts according to the sensor application defined by the ns-2 user. This application defines how a sensor will react once it detects its target phenomenon. For example, a sensor may periodically send a report to some data collection point as long as it continues to detect the phenomenon, or it may do something more sophisticated, such as collaborate with neighboring sensor nodes to more accurately characterize the phenomenon before alerting any outside observer of a supposed threat. For each sensor network there is a unique sensor application to accomplish threat detection, surveillance, environmental monitoring, etc. With ns-2, we have provided the facility to invoke sensor applications by phenomena. With these sensor applications, we can study how the underlying network infrastructure performs under tight energy, delay, and throughput constraints.

The following sections will document how we extended ns-2 to support simulating sensor networks, and we’ll illustrate its utility with a case study that examines the trade-offs between two routing algorithms for a rudimentary sensor application.

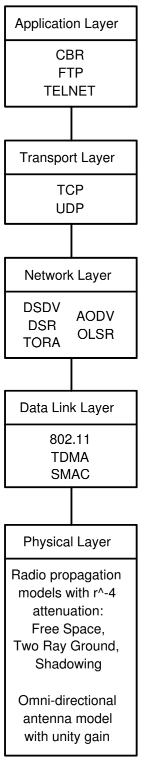

Application Layer
CBR
FTP
TELNET
Transport Layer
TCP
UDP
Network Layer
DSDV
DSR
TORA
AODV
OLSR
Data Link Layer
802.11
TDMA
SMAC
Physical Layer
Radio propagation models with r^-4 attenuation: Free Space, Two Ray Ground, Shadowing
Omni-directional antenna model with unity gain

Figure 1: These are some of the paradigms and protocols available for wireless networking in ns-2. Some protocols like OLSR [4] and SMAC [6] have not yet been incorporated into USC’s ns distributions, but they can be retrieved from their respective developers’ sites.

2 Related Work

Modeling sensor networks for simulation has not received as much attention from researchers as other areas like information processing and energy conservation. In fact, we are aware of only one project whose objective included building flexible simulation tools specifically for sensor networks. Park, Savvides, Srivastava [7] developed extensions to ns-2 for this purpose, but with an emphasis on sophisticated modeling of energy consumption and emulation (i.e. interfacing with real world sensor nodes). Unfortunately, their work has not been updated to support subsequent releases of ns-2 since October, 2000.

3 The Extended NS-2 Architecture

Figure 2 shows where our extensions are arranged within the ns-2 framework. The major additions and modifications are explained below. Section 3.1 shows how our extensions fit into ns-2's class hierarchy.

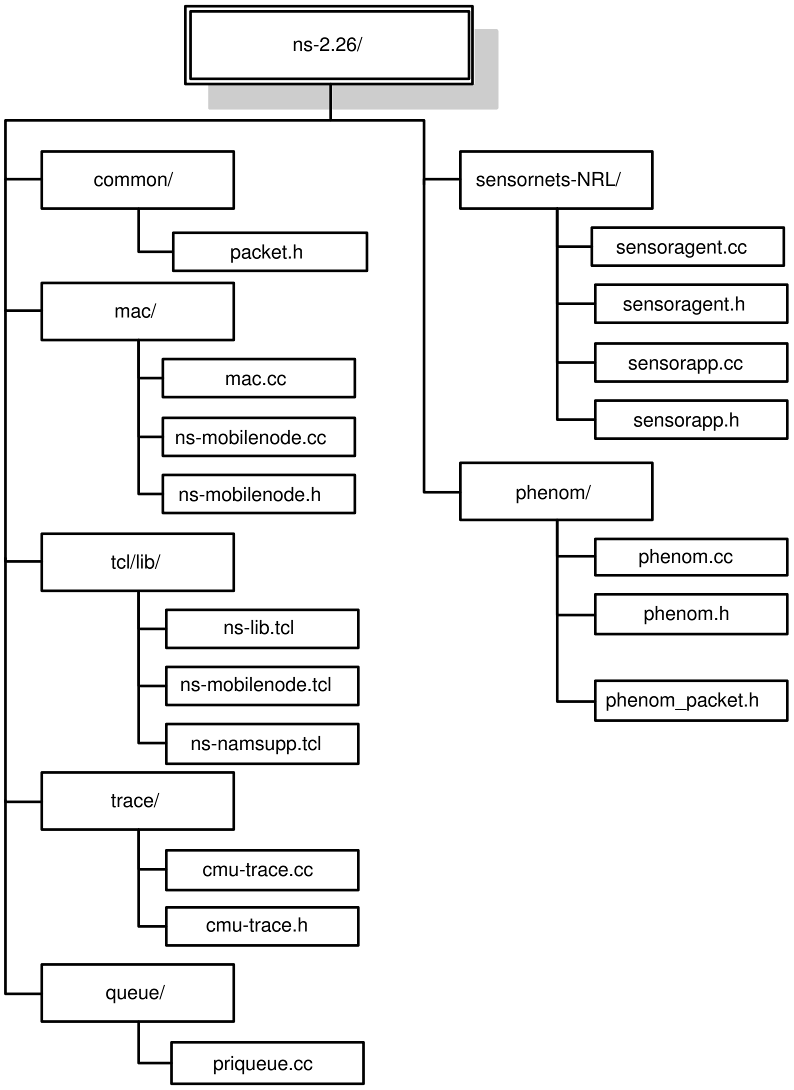


Figure 2: This figure illustrates which files in the ns framework were modified or added.

`trace/cmu-trace.cc` The `CMUTrace` class is used to print important parts of a packet to the simulation's trace file. Since we introduced a new packet type for phenomena, we had to add the corresponding format function in this class.

`tcl/lib/ns-lib.tcl` This component of the infrastructure interprets node configurations specified in the ns-2 simulation script. Our extensions introduced two new node types, the sensor node and the phenomenon node. Hence, we added some arguments in the `node-config` function to accommodate them.

`tcl/lib/ns-mobilenode.tcl` In the real world, phenomena usually don't emanate in an RF medium, but

In ns-2's virtual world, we're using its existing capacity for multi-channel wireless networking as a means to emanate phenomena of various kinds. By using a dedicated channel for phenomena, we can simulate the unique physical medium that that they occupy in the real world. Thus, sensor nodes will need to have two interfaces, one to the 802.11 channel and one to the PHENOM channel. We implemented this kind of "multi-homed" capability in the `add-PHENOMinterface` procedure of `ns-mobilnode.tcl`.

`common/packet.h` Each packet in ns-2 is associated with a unique type that associates it with the protocol that it belongs to, such as TCP, ARP, AODV, FTP, etc. Since we created a new protocol for phenomena, we defined it's corresponding packet type in the `packet.h` header file.

`mac/wireless-phy.cc` Ns-2 contains an energy model for wireless nodes which can be used to investigate the benefits of various energy conservation techniques, such as node sleeping or leveraging optimal network densities. The model includes attributes for specifying the power requirements of transmitting packets, receiving packets, or idly standing by during times of network inactivity. Sensing phenomena is a process that may consume power at another rate, so it's important to consider this where sensor network simulations are concerned. In `mac/wireless-phy.cc`, we've included the capability of specifying the amount of power consumed by nodes while sensing phenomena.

We modeled the presence of phenomena in ns-2 with broadcast packets transmitted through a designated channel. The range of phenomena is the set of nodes who can receive the PHENOM packets in that channel. This pattern will follow whichever radio propagation model (free space, two ray ground, or shadowing) included with the PHENOM node's configuration. These propagation models roughly cover a circle, but other shapes could be achieved by varying the range of PHENOM broadcast packets and creatively moving a set of PHENOM nodes emenating the same type of phenomenon.

Emenating PHENOM broadcast packets is accomplished by the "PHENOM routing protocol"[1], which simply broadcasts PHENOM packets with a certain configureable pulserate. When a PHENOM packet is received by a node listening on the PHENOM channel, a receive event is passed to that node's sensor application.

[1]This functionality best fit into ns-2's existing ad-hoc wireless networking infrastructure as a routing protocol, even though it does not route at all. The MAC layer it operates above must be specified in the PHENOM node's configuration. Although real-world phenomena can interfere in a variety of ways, we ignore this aspect and use non-interfering phenomena in order to reduce the randomness of traffic patterns and simplify the analysis of routing algorithms. So, in our simulations, we configure PHENOM nodes with the basic "Mac" class, which prevents channel contention.

Every sensor network simulation must have PHENOM nodes which trigger sensor nodes, but the traffic sensor nodes generate once they detect phenomena depends on the function of the sensor network. For example, sensor networks designed for energy efficient target tracking [8] would generate more sensor-to-sensor traffic than a sensor network designed to provide an outside observer with raw sensor data. This aspect of the simulation is defined by the sensor application, which is a modular component of the simulation environment intended to be customized according to the traffic properties associated with the sensor network being simulated.

`phenom/phenom.cc` This file implements the PHENOM routing protocol used for emenating phenommena. It includes parameters for the pulse rate and the phenomenon type (Carbon Monoxide, heavy seismic activity, light seismic activity, sound, or generic). These types are just names which can be used to identify multiple sources for phenomena in trace files. The pulse rate is the only parameter which actually controls how a PHENOM node emenates.

`sensornets-NRL/sensoragent.cc` The ns manual [2] describes *agents* as "endpoints where network-layer packets are constructed or consumed". Sensor nodes use a "sensor agent" attached to the PHENOM channel for consuming PHENOM packets, and a UDP or TCP agent attached to the wireless network channel for constructing packets sent down from the sensor application. Sensor agents act as a conduit through which PHENOM packets are received and processed by sensor applications. The sensor agent does not actually look at the contents of the PHENOM packet, it simply marks the packet as received and passes it to the sensor application. This agent is implemented in `sensoragent.cc`.

`sensornets-NRL/sensorapp.cc` The sensor application defined in this file utilizes node color and generates sensor reports to show when the corresponding sensor node detects phenomenon[2]. Specifically, when the node is receiving PHENOM packets, this application changes the node color to red, activates an "alarm" (public variable), and sends a "sensor report" of MESG_SIZE bytes to the sink node of a UDP (or TCP) connection once per TRANSMIT_FREQ seconds. When the node has not recieved a PHENOM packet in the timeout period specified by SILENT_PHENOMENON, then the node color changes back to green. If node color is desired to illustrate energy levels instead of sensor alarm status, then that aspect of the application can be disabled with DISABLE_COLORS.

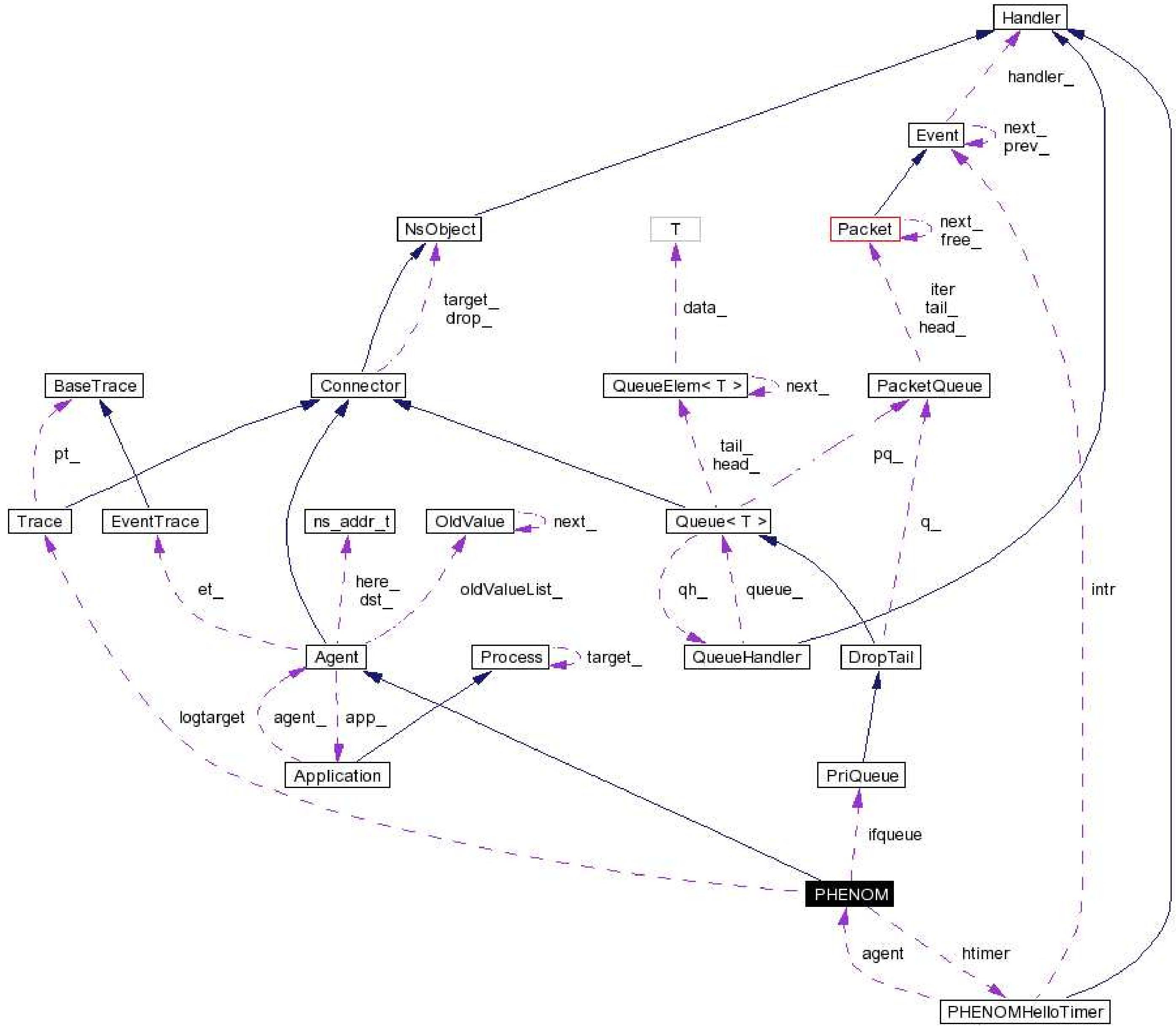
3.1 The Extended Ns-2 Class Hierarchy

Doxygen was used to generate the following figures which illustrate how our extensions to ns-2 fit into its class hierarchy.

4 Capabilities, Guidelines, and Caveats.

This section describes the capabilities of our sensor network extensions, gives some guidelines for configuring your own simulations, and attempts to explain some areas of likely confusion. In this section,

[2]The four environment variables which can be used to customize this application are SILENT_PHENOMENON, DISABLE_COLORS, MESG_SIZE, and TRANSMIT_FREQ. Anytime their values are changed, the sensorapp.o file must be rebuilt (just run `make` on the ns-2 Makefile).


Handler
handler_
Event
next_
prev_
NsObject
T
Packet
next_
free_
target_
drop_
data_
iter
tail_
head_
BaseTrace
Connector
QueueElem< T >
next_
PacketQueue
pt_
tail_
head_
pq_
Trace
EventTrace
ns_addr_t
OldValue
next_
Queue< T >
q_
intr
et_
here_
dst_
oldValueList_
qh_
queue_
Agent
Process
target_
QueueHandler
DropTail
logtarget
agent_
app_
Application
PriQueue
ifqueue
PHENOM
agent
htimer
PHENOMHelloTimer

we assume the reader is already familiar with setting up mobile node simulations in ns-2. For readers who are not, you will probably have difficulty following along until you have done some basic ns-2 wireless simulations. You may find the necessary background at the following URLs:

```
http://nile.wpi.edu/NS/
http://www.isi.edu/nsnam/ns/tutorial/index.html
http://www.isi.edu/nsnam/ns/tutorial/nsscript5.html
```

The easiest way to create sensor network simulations is to use the `script_maker.pl` utility in the `simulations_aids` directory distributed with our extensions. This Perl script contains commonly used parameters for setting up sensor network simulations and automatically generates the (often) complex NS simulation script. The remainder of this section describes how to code a sensor network simulation into the NS simulation script, without using the `script_maker.pl` utility.

Setting up a sensor network in ns-2 follows the same format as mobile node simulations. The best way to create your own simulation is to modify the one of the examples distributed with this code.

Places where a sensor network simulation differs from a mobile node simulation are listed below. Setting up `ns_`, `god_`, tracing, topography objects and starting and stopping the simulation are all the same as in traditional mobile node simulations.

- Configure phenomenon channel and data channel.

  Create a phenomenon channel and a data channel. Like mobile nodes, phenomenon nodes use 802.11 for the physical layer. We must configure the two different types of nodes on separate channels, in order to avoid contention at the physical layer. All phenomenon nodes should be configured on the same channel, even if they're emanating different types of phenomena.

  ```
  set chan_1_ [new $val(chan)]
  set chan_2_ [new $val(chan)]
  ```

- Choose a MAC layer to use for emanating phenomena over the existing ns-2 infrastructure for wireless communications. Using 802.11 probably isn't realistic, since phenomena should be emanating without regard to collisions or congestion control (i.e. do we really need to simulate collisions of atomic molecules, such as carbon monoxide? Even if we did, I doubt 802.11 would model that very well...). I recommend using the basic "Mac" class for the PHENOM node's MAC layer, via:

  ```
  set val(mac)        Mac/802_11  ;# MAC type for sensor nodes
  set val(PHENOMmac)  Mac         ;# MAC type for phenomena
  ```

- Configure phenomenon nodes with the PHENOM "routing" protocol:

  Use node-config, just like with mobile nodes, but specify PHENOM as the routing protocol so the phenomenon is emanated according to the rules defined in phenom/phenom.cc. Also, be sure to configure in the channel and MAC layer you've selected for phenomena broadcasts. In this example, we selected `$chan_1_`.

  ```
  $ns_ node-config \
  ```

```
    -adhocRouting PHENOM \
    -channel $chan_1_ \
    -llType $val(ll) \
    -macType $val(PHENOMmac) \
    -ifqType $val(ifq) \
    -ifqLen $val(ifqlen) \
    -antType $val(ant) \
    -propType $val(prop) \
    -phyType $val(netif) \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace ON \
    -movementTrace ON
```

- Configure the Phenomenon node's pulse rate and phenomenon type.

  The two parameters which can be used to customize the Phenomenon are listed below. They are both optional.

  1. pulserate FLOAT
     - FLOAT must be either a floating point number or an integer
     - describes how frequently a Phenomenon node broadcasts its presence
     - defaults to 1 broadcast per second
  2. phenomenon PATTERN
     - PATTERN must be any one of the following keywords: CO, HEAVY_GEO, LIGHT_GEO, SOUND, TEST_PHENOMENON corresponding to Carbon Monoxide, heavy seismic activity, light seismic activity, audible sound, and some other generic phenomenon.
     - This option is mostly useful for simulations involving multiple phenomenon nodes, so that it's easier to distinguish who a sensor node is detecting by looking at the NS trace file.
     - defaults to TEST_PHENOMENON

       ```
       [$node_(0) set ragent_] pulserate .1 ;#PHENOM emanates 10x/s
       [$node_(0) set ragent_] phenomenon CO ;#Carbon Monoxide PHENOM
       ```

- Configure sensor nodes.

  Sensor nodes must be configured with the `-PHENOMchannel` attribute and the `-channel` attribute. `PHENOMchannel` should be the same as the channel you configured the phenomenon node with. The other channel is the channel which will be used for mobile-node application layer data, such as sensor reports. Also, sensor node configurations must specify a MAC protocol for the phenomena channel and a MAC protocol (such as `Mac/802_11`) for the channel shared with other wireless nodes. Do this with the `-PHENOMmacType` and `-macType` attributes. `PHENOMmacType` should be the same as the macType used in PHENOM nodes, and `macType` should be the same as the macType used in other nodes participating in the processing of sensor data (probably 802.11).

  ```
  $ns_ node-config \
  ```

```
-adhocRouting $val(rp) \
-channel $chan_2_ \
-macType $val(mac) \
-PHENOMmacType $val(PHENOMmac) \
-PHENOMchannel $chan_1_
```

If desired, a sensor node can be configured so that a specified amount of energy will be deducted from a sensor node's energy reserve each time it receives any amount of phenomena at some point in time. To set this up, the following parameters must be used in node-config:

```
-energyModel $val(engmodel) \
-rxPower $val(rxPower) \
-txPower $val(txPower) \
-sensePower $val(sensePower) \
-idlePower $val(idlePower) \
-initialEnergy $val(initeng)
```

Each of those values must be previously initialized, such as:

```
set val(engmodel)   EnergyModel
set val(txPower)    0.175       ;# transmitting power in mW
set val(rxPower)    0.175       ;# receiving power in mW
set val(sensePower) 0.00000175; ;# sensing power in mW
set val(idlePower)  0.0         ;# idle power in mW
set val(initeng)    0.5         ;# Initial energy in Joules
```

The energy parameters used in node-config are described as follows:

- `rxPower .175` ← indicates $175mW$ consumed for receiving a packet of arbitrary size at time t
- `txPower .175` ← indicates $175mW$ consumed for transmitting a packet of arbitrary size at time t
- `sensePower .00000175` ← indicates $1.75\mu W$ consumed for detecting any amount of phenomena at time t
- `initialEnergy 5` ← indicates a total energy reserve of $5J$ available to the sensor

IMPORTANT CAVEAT:

ns-2's energy consumption model utilizes color to illustrate when a node is about to exhaust it's energy, so the node coloring which is part of the sensor app should be disabled with the `DISABLE_COLORS` definition in `sensorapp.cc`. Remember to run make again to compile those changes into the NS executable.

In addition to `DISABLE_COLORS`, some other sensor node parameters can be specified in `sensorapp.cc`. These parameters are listed below:

SILENT_PHENOMENON is the seconds of quiescence required for a sensor to go off it's alarming state. Example:

```
#define SILENT_PHENOMENON 0.2
```

DISABLE_COLORS disables node color changes invoked by the sensor application. This is useful when it is desired to use node color to illustrate a node's energy reserves. Example:

```
#define DISABLE_COLORS FALSE
```

MESG_SIZE is the size (in bytes) of the messages to send to the gateway, or data collection point, or whatever you want to call the sink node attached to this sensor node (over UDP, for example). Example:

```
#d#define MESG_SIZE 1000
```

TRANSMIT_FREQ is the frequency with which a sensor node triggered by PHENOM pkts will send a message to the gateway, or data collection point, or sink, or whatever you want to call the sink node attached to this sensor node (over UDP, for example). Units are in seconds, so a message of size `MESG_SIZE` bytes will be transmitted to the gateway node once for every `TRANSMIT_FREQ` seconds in which the sensor node has received one or more PHENOM packets.

```
#define TRANSMIT_FREQ 0.1
```

- Configure non-Sensor nodes (such as data collection points, or gateways for the sensor network).

  Nodes which are not sensor nodes or phenomenon nodes, should not be configured with a `PHENOMchannel`, since their only interface is to the mobile-node channel. This is done with the `-PHENOMchannel "off"` attribute.

```
$ns_ node-config \
   -adhocRouting $val(rp) \
   -channel $chan_2_ \
   -PHENOMchannel "off"
```

- Attach sensor agents.

  Create a sensor agent for each sensor node, and attach that agent to its respective node. Also, specify that all packets coming in from the PHENOM channel should be received by the sensor agent. In the following example, `$i` would represent the node number for the sensor node currently being configured.

```
set sensor_($i) [new Agent/SensorAgent]
$ns_ attach-agent $node_($i) $sensor_($i)

# specify the sensor agent as the up-target for the sensor node's link
# layer configured on the PHENOM interface, so that the sensor agent
# handles the received PHENOM packets instead of any other agent
# attached to the node.
[$node_($i) set ll_(1)] up-target $sensor_($i)
```

- Attach UDP agent and sensor application to each node (optional).

  How the sensor nodes react once they detect their target phenomenon is a behavior which should be defined as a sensor application. One such application might involve sensor nodes alerting a data collection point via UDP with information about the phenomenon. The following example illustrates how an application like that would get setup. Again, `$i` would represent the node number for the sensor node currently being configured.

  ```
  set src_($i) [new Agent/UDP]
  $ns_ attach-agent $node_($i) $src_($i)
  $ns_ connect $src_($i) $sink

  set app_($i) [new Application/SensorApp]
  $app_($i) attach-agent $src_($i)
  ```

- Start the phenomenon node.

  The Phenomenon node starts emanating immediately once the simulation starts, but the range of it's broadcasts can be reduced to such a small area that it's effectively inaudible to any sensors (unless they occupy the exact same coordinate in the grid). Here's an example of how a phenomenon node can be turned off:

  ```
      $ns_ at 6.0 {[$node_($i) set netif_(0)] set Pt_ 0.0001}
  ```

  `Pt_` is the range of the broadcast, and `$i` is the node id of the Phenomenon node.

- Start the sensor application.

  The sensor node will receive PHENOM packets as soon as the sensor agent is attached to the node. Since the sensor agent does nothing but notify the sensor application of the event, ”I received PHENOM“, the sensor node does not visibly react to PHENOM packets until the sensor application has been attached and *started*. The following example shows how to start a sensor application:

  ```
      $ns_ at 5.0 "$app_($i) start $sensor_($i)"
  ```

5 Bugs

one bug: phenom nodes receive bcasts from other phenom nodes... not realistic, but doesn't seem to effect simulation results on the IP side of the sensor network. But, it does make the simulations much longer and trace files much bigger when multiple phenom nodes are being used in close proximity.

No bugs are presently known with the sensor network extensions to ns-2, but they no doubt exist. Please direct all bug reports to Ian Downard, `<downard@itd.nrl.navy.mil>`.

6 Future Work

Much more effort should be made to improve how phenomenon emanates. Presently, it follows the behavior of an 802.11 broadcast, configured with one of the following radio propagation models:

1. Free Space Model
2. Two Ray Ground Model
3. Shadowing Model

The first two models represent the communication range as an ideal circle, whose boundary is an absolute limit on signal receivability. The Shadowing model applies a more probabilistic means of determining whether a receiver on the boundary can receive the signal.

The radio propagation model should be extended to create a "phenomenon propagation model" which could specifically address the characteristics of various phenomenon available for sensor network simulations.

7 Case Study: MANET Routing Within a Dynamic Sensor Network

This case study begins to show the types of results one can achieve from sensor network simulations with ns-2.

8 Software References

8.1 NRL Sensor Network Extensions to NS-2

The NRL SensorSim extensions to NS–2 facilitate simulating sensor networks. The code and documentation for extending NS–2.1b9a or NS–2.26 is available here:

`http://pf.itd.nrl.navy.mil/projects/nrlsensorsim/`

8.2 NRL OLSR Extensions to NS-2

One of the original motivations behind building the sensor network extensions into ns-2 was to compare the behaviors of OLSR and AODV routing algorithms. NRL's OLSR extension to NS–2 is available for download via CVS [2]. To check out all the nrlolsr files (from a Linux box), type the following two commands:

```
cvs -d :pserver:anonymous@nrlolsr.pf.itd.nrl.navy.mil:/cvsroot/nrlolsr login
cvs -z3 -d :pserver:anonymous@nrlolsr.pf.itd.nrl.navy.mil:/cvsroot/nrlolsr co .
```

The instructions for including nrlolsr extensions in ns-2 are documented in nrlolsr/readme. To get the code, follow the nrlolsr-NS- link at this page:

`http://pf.itd.nrl.navy.mil/projects/olsr/`

References

[1] The Network Simulator - ns-2, `http://www.isi.edu/nsnam/ns/`

[2] The ns Manual, `http://www.isi.edu/nsnam/ns/ns-documentation.html`

[3] NRL's Sensor Network Extension to ns-2, `http://nrlsensorsim.pf.itd.nrl.navy.mil/`

[4] NRL's OLSR implementation for ns-2, `http://pf.itd.nrl.navy.mil/projects/olsr/`

[5] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva. "A performance of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 1998.

[6] Wei Ye, John Heidemann, Deborah Estrin. "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," in proceedings of the IEEE INFOCOM, 2002.

[7] SensorSim: A Simulation Framework for Sensor Networks. `http://nesl.ee.ucla.edu/projects/sensorsim/`

[8] H. Yang, B. Sikdar. "A Protocol for Tracking Mobile Targets using Sensor Networks," `Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications`, pp. 71-81, Anchorage, AK, May 2003.